On Cost-effective Reuse of Components in the Design of Complex Reconfigurable Systems

J. I Aizpurua*, Y. Papadopoulos†, E. Muxika‡, F. Chiacchio⋆, G. Manno°

*Institute for Energy and Environment, Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, UK

†Computer Science Department, University of Hull, Hull, UK

‡Electronics and Computing Department, Mondragon University, Arrasate, Spain

⋆University of Catania, Department of Mathematics and Informatics, Catania, Italy

°DNV GL, Digital Solutions & Innovation, Høvik, Norway

Abstract: Design strategies that benefit from the reuse of system components can reduce costs whilst maintaining or increasing dependability—we use the term dependability to tie together reliability and availability. D3H2 (aDaptive Dependable Design for systems with Homogeneous and Heterogeneous redundancies) is a methodology that supports the design of complex systems with a focus on reconfiguration and component reuse. D3H2 systematises the identification of heterogeneous redundancies and optimises the design of fault detection and reconfiguration mechanisms, by enabling the analysis of design alternatives with respect to dependability and cost. In this paper, we extend D3H2 for application to repairable systems. The method is extended with analysis capabilities allowing dependability assessment of complex reconfigurable systems. Analysed scenarios include time-dependencies between failure events and the corresponding reconfiguration actions. We demonstrate how D3H2 can support decisions about fault detection and reconfiguration that seek to improve dependability whilst reducing costs via application to a realistic railway case study.

Keywords: Dynamic dependability, repairable systems, reconfigurable systems, heterogeneous redundancies, cost-effectiveness, design methodology, adaptive systems.

1 Introduction

The improvement of system dependability and the reduction of cost are typically competing goals in the design of systems[1,2]. Improvement of dependability is often achieved via use of fault tolerance.

Traditional design strategies for improving fault tolerance are based on the replication of hardware components in redundant configurations, for instance primary/standby or triple modular redundancy[3]. Hardware replicas perform identical functions and accordingly they are known as homogeneous redundancies. Combined with design diversity, this can be an effective strategy for improving dependability. However, the replication of hardware resources often results in unnecessary additional costs.

In highly networked scenarios there is room to take advantage of over-dimensioning design decisions and overlapping functions by exploiting heterogeneous redundancies, i.e. components that, besides performing their primary intended design function, can also be used as a means of restoring the functionalities lost when other components fail[4–7]. For systems with high dependability requirements the effects of such use on dependability must be established. While dependability integrates different attributes[8], in this paper we focus on reliability and availability.

Highly networked scenarios comprise of many processing units, sensors, and actuators connected to a communication network with the particularity that replicas of system functions are distributed throughout the physical structure. For example, trains have replicated functions throughout their cars and large buildings have replicated control functions throughout their floors and rooms. Assuming that such heterogeneous redundancies exist and can be exploited in case of failures, the system must include fault detection and reconfiguration implementations (i.e., health management mechanisms) that deploy these redundancies. Heterogeneous redundancies typically operate as cold-standby redundancies which need to be activated in the presence of failures[7]. Failing to activate a redundancy has consequences for dependability which must be established. Different decisions about use of heterogeneous redundancies yield different dependability and cost values for a system and, therefore, evaluation of design options is needed to arrive at a decision that can achieve high dependability with acceptable costs.

To systematize and integrate these concepts in a method for the assessment and design of complex reconfigurable systems, we have created the D3H2 (aDaptive Dependable Design for systems with Homogeneous and Heterogeneous redundancies) methodology[4,6]. The aim of D3H2 is to identify heterogeneous redundancies; create architectures that exploit homogeneous/heterogeneous redundancies; and evaluate the influence of design decisions on dependability and cost. D3H2 provides the data that supports trade-off design decisions between dependability and cost when deciding to implement different types of redundancy and health management strategies.

In previous work, we have developed D3H2 for a class of non-repairable systems[5,7]. However, most complex industrial systems can be considered repairable[9]. We have, therefore, extended the D3H2 methodology to cover repairable systems. One challenge is that, for non-repairable systems only the order of failure is important, but for repairable systems both the order of failure and repair must be respected. A key innovation in D3H2 is that the reconfiguration process is governed by the reconfiguration priority of implementations. This means that the failure/repair reconfiguration is not necessarily defined *a priori*, but it can follow a dynamically decided pattern. For instance, assume that there are four implementations ordered with their priority and currently the third implementation is operative while first and second implementations have failed (cf. Figure 1, $t=t_3$). If the first or second implementation is repaired before the third fails, when the third implementation fails the first or second implementation should be activated instead of the fourth implementation.

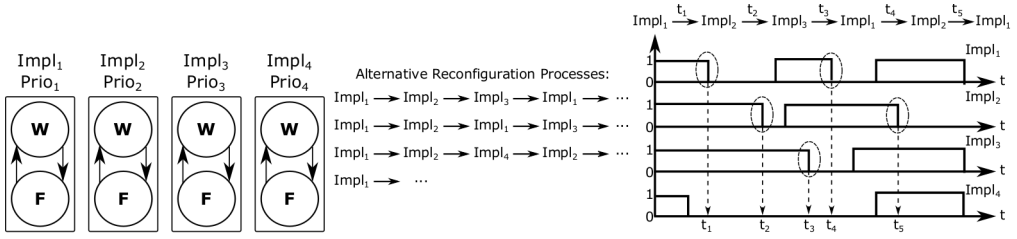

Figure 1: Possible random reconfiguration sequences.

This type of complex repair process cannot be modelled with existing dynamic dependability formalisms such as Dynamic Fault Trees [10] because their modelling constructs assume fixed sequences and they are insufficient for capturing this complex repair pattern. Although in theory it is possible to use low-level pure stochastic models (e.g., Markov chains), their effectiveness for complex models is limited because it is difficult to trace from the design model to the analysis model and their size growths rapidly leading eventually to state-explosion problems [11].

The main contribution of this paper is thus the extension of the D3H2 methodology to enable the design of repairable systems, which include complex failure and repair event sequences. The methodology encompasses the implementation of user-defined reconfiguration strategies and the systematic evaluation of the influence on dependability of design decisions including redundancy strategies and health management mechanisms. The second contribution of the paper is the application of the D3H2 methodology for the evaluation of the reuse of repairable components in a railway case-study.

The remainder of this paper is organised as follows. Section 2 reviews the relevant work, Section 3 introduces the D3H2 methodology, Section 4 presents the running case study, Section 5 specifies in detail the system design, Section 6 describes the dependability evaluation approach, and, finally, Section 7 presents conclusions and future prospects.

2 Related Work

The design of reconfigurable systems is an ongoing research challenge. While many works have concentrated on analysing the influence of homogeneous redundancies [12–16], approaches focusing on the evaluation of heterogeneous redundancies are scarce [7,10]. Heterogeneous redundancies can take many forms: design diversity [17], analytical redundancies [18], or redundancies arising from overlapped system functions [4].

In our approach we focus on identifying and exploiting implicit redundancy which may exist in an application. Detailed knowledge and mathematical formulation of the system is typically needed to get analytical redundancy relations [18]. However, the complexity of the mathematical formulation increases with the system size, and this has led us to adopt a function-based viewpoint that uses qualitative attributes (see also Subsection 5.2). The use of functional alternatives to compensate for component failures is discussed in [19]. The authors use weighted sums to to combine different attributes and compare

the overall utility of alternative configurations. The shared redundancy concept is presented in[20] with the goal of reusing processing units in the presence of software component failures. Authors perform availability and cost evaluations using Fault Trees and Monte Carlo simulations. Implicit redundancies are also aligned with the goal of reusing components[21]. The paper describes an adaptation model used to specify for each component its implicit redundancies and quality constraints. Component Fault Trees and Markov chains are used to estimate failure probabilities. Similarly, the integrated modular avionics paradigm shares the goal of replacing software units via standardized generic hardware modules[22]. Their goal is not to use heterogeneous redundancies in highly networked scenarios, but exploit replaceable processing units in reconfiguration.

While the influence of fault detection, reconfiguration and communication implementations on system design has been addressed for homogeneous redundancies, to the best of our knowledge, these mechanisms has been assumed ideal for heterogeneous redundancies. The evaluation of the faulty behaviour of these implementations leads to obtaining an approach which better adheres to reality and consequently provides more accurate estimation of dependability. In D3H2, dependability is a key criterion of performance in the decision between alternative reconfiguration strategies. Due to the complex, dynamic and repairable nature of the systems, we need a dependability approach which is able to specify:

(S1) Time-dependent behaviour of system configurations.

(S2) Modular or hierarchical system failure behaviour to manage the complexity of the model and be able to trace from the design model to the dependability model and vice-versa.

(S3) Repair behaviour of hardware, software and communication resources of the system.

(S4) Any cumulative distribution function for failure and repair events.

(S5) User-defined reconfiguration strategies according to the defined configuration priorities.

There is a wealth of recent development in dependability analysis from which D3H2 could benefit. Dynamic Fault Trees (DFT) extend Fault Trees to integrate system dynamics[23]. Dynamic Fault Trees have been extended to address repairable systems by embedding repair mechanisms in the failure specification logic[24]. Similarly, Dynamic Reliability Block Diagrams (DRBD) are based on the dynamic extension of Reliability Block Diagrams[25]. In DRBD each block is modelled with three possible states: operating, standby, and failed state. Transitions between these states are defined with four events: wake-up, sleep, repair, and failure. For the dependability assessment the DRBD approach defines cause-effect relationships between connected blocks. HiP-HOPS[26] is a modular dependability analysis approach which integrates dynamic analysis with design optimization and safety requirement allocation using meta-heuristics. The designer makes failure annotations in the design model and HiP-HOPS synthesizes Dynamic Fault Trees used for subsequent analysis and optimisation of the system design[27,28]. Boolean Driven Markov Processes (BDMP)[29] integrate Markov chains and Fault Trees to specify the

dynamic failure behaviour. In a BDMP model, different events (or leafs) can trigger other events in the Fault Tree dynamically. The specification of leafs is done with predefined Markov chains.

The modular system failure specification has been addressed for different dynamic dependability models such as Dynamic Fault Trees[30,31]. Other dependability analysis approaches integrate the modular specification logic in the dependability specification formalism through the transformation of a high-level component-based model into a low-level dependability analysis formalism for quantification. State-Event Fault Trees (SEFT)[32] combine the specification of Component Fault Trees with state-machine representations in order to specify the failure behaviour of repairable systems in a modular way. In order to quantify the SEFT model, it is transformed into an underlying Deterministic and Stochastic Petri nets model. Similarly, Generalized Fault Trees (GFT)[33] rely on transformations to solve high-level GFT models which combine parametric and repairable DFT concepts. As for the quantification of Generalized Fault Tree models, they are transformed into Stochastic Well-Formed Nets. Although these top-level formalisms are modular, their transformation into a low-level formalism results in a flat dependability analysis model. Table 1 displays analysed dynamic dependability analysis techniques and addressed properties.

Table 1: Dynamic dependability approaches and specification capabilities.

Approach	(S1) Temporal	(S2) Modular	(S3) Repair	(S4) Any CDF	(S5) Reconfiguration
DFT[24]	✓	✓	✓	✓	X
DRBD[25]	✓	✓	✓	✓	X
HiP-HOPS[26]	✓	✓	X	✓	X
BDMP[29]	✓	✓	✓	✓	X
SEFT[32]	✓	✓	✓	X	X
GFT[33]	✓	✓	✓	✓	X

Most approaches in Table 1 address temporal analysis, can be applied in a modular fashion, can deal with repair, and assume any cumulated distribution function for component failures. However, approaches to repair require users to make *a priori* assumptions about this repair process which have a static character. For instance, DFT spare gates require predefined repair priorities[24], DRBD embeds possible dependencies[25], and BDMP defines the reactivation logic for inter-dependent components with predefined trigger mechanisms[29]. Fixing elements of the repair logic, however, has its drawbacks; for instance, it is difficult, if not impossible, to represent situations where repair is dynamically decided. Although a few techniques have been extended with more flexible mechanisms (e.g., BDMP[34]), the representation and analysis of dynamic repair scenarios remains a research challenge that we try to address within D3H2.

3 Overview of the D3H2 Methodology

D3H2 integrates the modelling and analysis activities as shown in Figure 2. Systems are specified as a set of interacting hardware, software, and communication resources, including their interfaces and provided functionality.

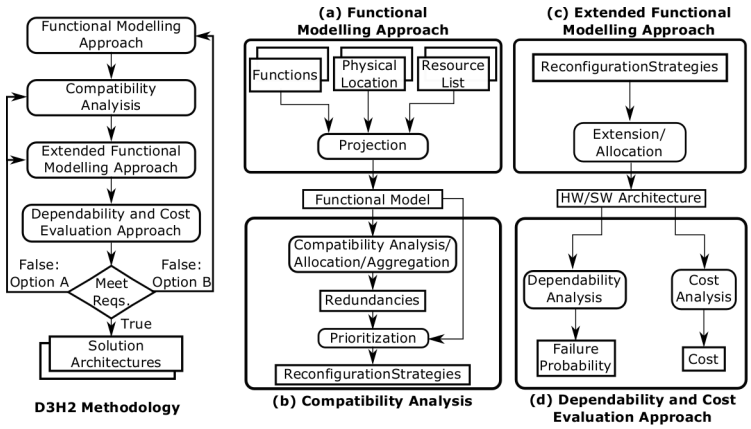

Figure 2: D3H2 design methodology[4].

The main approaches integrated in the D3H2 methodology are listed below:

- The Functional Modelling Approach specifies the functional model including system functions and related attributes including the physical location in which these functions are performed and a necessary list of resources to develop these functions (see Subsection 5.1).

- The Compatibility Analysis identifies compatible implementations (i.e., redundancies) in the functional model. To use these compatible implementations, it may be necessary to aggregate additional resources and perform reallocation of new elements. Subsequently, reconfiguration strategies and reconfiguration priorities are defined (see Subsection 5.2).

- The Extended Functional Modelling Approach (see Subsection 5.3) revisits the functional model to include the fault detection and reconfiguration functions needed to implement the strategies identified in Compatibility Analysis. The functional model is also extended to include allocation of hardware/software (HW/SW) resources to the system functions. At this point, a HW/SW architecture emerges and the effect of design improvements on dependability and cost can be assessed.

- The Dependability and Cost Evaluation Approach predicts the dependability and cost of the HW/SW architecture. Via iterative application and comparison of results, it enables the adoption of informed trade-off decisions between candidate design decisions and incurred cost (see Section 6). The HW/SW architecture needs to be evaluated to verify if the initial requirements are

met. If they are not satisfied there are two options: Option A takes the process to an earlier activity and iterates from there while Option B moves the design process back to its starting point so that design requirements are reconsidered. Depending on the requirements, Option A redirects the design flow to an intermediate design step: redundancy-related design decisions are reconsidered through the application of the Compatibility Analysis (e.g. changing homogeneous redundancies with heterogeneous redundancies to reduce design costs), whereas health management functions are reconsidered through the Extended Functional Modelling Approach (e.g. reducing fault detection implementation redundancies to reduce design costs). Generally the application of the Compatibility Analysis implies the application of the Extended Functional Modelling Approach. The reconsideration of design requirements from Option B results in the redesign of the functional model. Note that the fault hypothesis that underpin the dependability analysis in the D3H2 is the occurrence of permanent, but potentially repairable, dynamic failures of hardware, software, and communication components which are manifested with loss of function (omission failure) or delivery of function out of context (commission failure)[8].

The four approaches of D3H2 will be discussed with the aid of a railway system which is introduced next and described in more detail in[6].

4 Train Car Door Status Control System

The door status control is a safety-critical function which determines the safe operation of door open and close actions. It has dependencies with other systems of the train and the door operations are controlled by the driver depending on the status of the train, e.g. the doors must remain closed while the train is running. Each door in the train has sensors and control buttons for the passengers and the driver. Figure 3 shows the door status control configuration including both hardware and logical dependency models. There is one opening and closing button for the driver connected to the processing unit of the driver (PU_{Driver}) and each door throughout the train has: one opening button for passengers, one door speed sensor, one door open detection sensor, one door closed detection sensor and one obstacle detection sensor. All these sensors, their controllers, and the door control algorithm are located in the processing unit PU_{Door}.

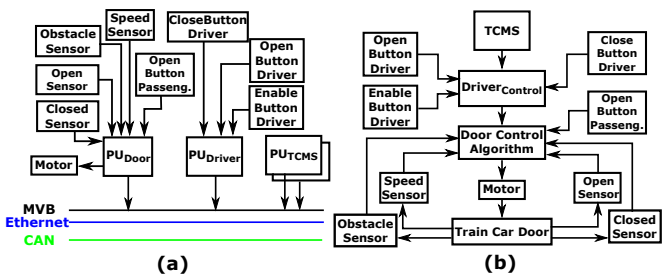

Figure 3: Door status control: (a) hardware dependencies and (b) logical dependencies.

In the train there is a component called TCMS (Train Control and Monitoring System), which monitors and controls different critical systems of the train such as traction and doors. This component is homogeneously duplicated in two reliable processing units (PU_{TCMS}) for safety purposes. The TCMS receives information about the speed of the train and it will not allow the driver to open the doors while the train is running. To this end, the TCMS sends an enable signal to the driver to inform about the safe operation of door opening or closing (Enable Door Driver - EDD). Using the information of the Enable Door Driver signal, the driver sends an enable signal to the controller of each door (Enable Door Passenger - EDP) to act safely on opening/closing the doors, while taking into account if the train is moving and if there is an obstacle in the door (cf. Figure 3b). All the processing units of the door status control system are connected to Multifunction Vehicle Bus (MVB)[35]. Other systems in the train are connected to Ethernet (e.g., video surveillance) and CAN (e.g., fire protection) communication networks. An interconnecting gateway enables the communication between processing units connected to different communication networks.

5 System Design using D3H2

5.1 Functional Modelling Approach

The Functional Modelling Approach specifies the functional operation of the system in a top-down manner. Inspired from SADT (Structured Analysis & Design Technique)[36], a set of tokens aid in the systematic specification of the key operational parts of the system starting from a set of high-level functions (e.g., different railway train operations: train operating properly, train stopped) tracing down to the necessary resources to perform these functions:

- A high level function consists of a set of Main Functions (MF), e.g., train operating properly = {traction system OK, signalling system OK, braking system OK, air conditioning control OK, ...}.
- Main functions are performed in possibly different Physical Locations (PLs), e.g., a single air conditioning control implementation may span a whole train car or each car compartment in a train car may have its own air conditioning control.
- A main function consists of a set of subfunctions (SF), e.g., input, control and output subfunctions.
- A subfunction may have multiple implementations (#) to carry out the subfunction and these are ordered with respect to their priority.
- Each implementation requires a set of hardware, software and communication resources.

For simplicity, the token-based specification process focuses on main functions and a first level of decomposition from main functions to subfunctions. However, the Functional Modelling Approach is

extendible to N functional levels. The full specification of a subfunction's implementation of a generic main function is specified as follows:

$$\texttt{Main Function.Physical Location.Subfunction.Implementation} \quad (1)$$

To define the physical location of system functions consistently, a physical location map is defined for the physical structure. Figure 4 shows the physical location map of an hypothetical train, where each car of the train is comprised of different compartments (Zone_A, Zone_B).

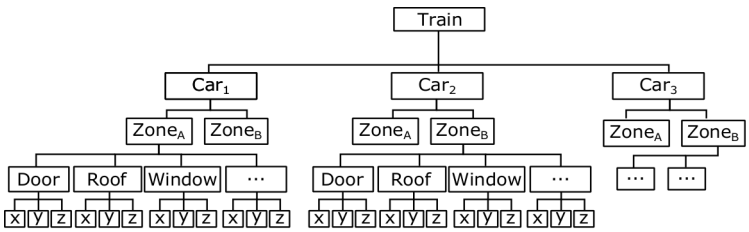

Figure 4: Physical location map[6].

Based on the token-based specification defined in Eq. (1), Table 2 describes the functional model of the door status control (cf. Figure 3).

Table 2: Functional models of door status control and video surveillance.

Main Function	**Physical Loc.**	**Subfunction**	**Resources**	**#**
Door Status Control	Train. Car_1. Zone_A. Door	Enable Door Driver (EDD)	PU_{TCMS1}, SW_{TCMS1}	1
		Enable Door Driver (EDD)	PU_{TCMS2}, SW_{TCMS2}	2
		Enable Door Passenger (EDP)	EDD, $\text{PU}_{\text{Driver}}$, EnableButton, Communication	3
		Door Close Command (DCC)	$\text{PU}_{\text{Driver}}$, $\text{CloseButton}_{\text{Driver}}$	4
		Door Open Command (DOC)	$\text{PU}_{\text{Driver}}$, $\text{OpenButton}_{\text{Driver}}$	5
		Door Open Command (DOC)	PU_{Door}, $\text{OpenButton}_{\text{Passenger}}$	6
		Door Open Detection (DOD)	PU_{Door}, OpenSensor	7
		Door Closed Detection (DCD)	PU_{Door}, ClosedSensor	8
		Door Velocity (DV)	PU_{Door}, VelocitySensor	9
		Obstacle Detection (OD)	PU_{Door}, ObstacleSensor	10
		Door Control Algorithm (DCA)	EDP, DCC, DOC, DOD, DCD, DV, OD, PU_{Door}, SW_{DSC}, Communication	11
		Door Manipulation (DM)	DCA, PU_{Door}, Motor	12
Video Surveillance	Train. Car_1. Zone_A. Door	Video Input	Camera, PU_{Cam}	13
		Process Image	Video Input, $\text{SW}_{\text{Surveillance}}$, PU_{Cam}	14
		Alarm	Process Image, Siren_A	15

The door status control main function requires different input subfunctions to assure the safe operation of door opening/closing: enable subfunctions (enable door driver — EDD [#1, #2], enable door passenger — EDP [#3]), command subfunctions (door close command — DCC [#4], door open command - DOC [#5, #6]), and monitoring subfunctions (door open detection — DOD [#7], door closed detection — DCD [#8], door velocity — DV [#9], obstacle detection — OD [#10]). Door open commands are generated by passengers and the driver, but the door close command is controlled only by the driver. These input subfunctions are directed toward the door control algorithm (DCA) subfunction [#11] which determines when and how to close the doors through the door manipulation (DM) subfunction [#12]. Note that the final decision on opening/closing the door relies on the Enable Door Passenger (EDP) signal, which is determined by the driver.

Table 2 also shows the functional model of the video surveillance main function, which is connected to the Ethernet communication network and it is located in the same physical location as the door status control main function (cf. Figure 4): Train.Car_1.Zone_A.Door. The video surveillance function receives video images via video input subfunction [#13], processes them through the process image subfunction [#14] and, when conditions are met, it raises an alarm using the sirens connected to the PU_Cam [#15].

5.2 Compatibility Analysis

The Compatibility Analysis identifies heterogeneous redundancies based on tokens of the functional model (cf. Eq. (1)). There may exist two compatibility cases among the system implementations defined in the functional model:

- Natural compatibility is the case of implementations carrying out the same subfunction in compatible physical locations.
- Forced compatibility is the case of implementations carrying out different but potentially equivalent subfunctions located at compatible physical locations.

To identify heterogeneous redundancies we identify matching subfunctions and compatible physical locations in the functional model to determine if the analysed implementations are compatible or not. We define compatible physical locations according to the location of subfunctions (cf. Figure 4): (1) same physical location; (2) adjacent physical locations ([Train].[Car_1].Zone_A $\leftrightarrow$ [Train].[Car_1].Zone_B); or (3) physical locations that span other physical locations ([Train].[Car_1].[Zone_A] $\rightarrow$ [Train].[Car_1].[Zone_A].Door). Focusing on forced compatibilities we can see that the door status control and video surveillance main functions in Table 2 are located in a compatible physical location. Based on engineering design knowledge, we can identify that the video surveillance can provide a compatible implementation to the door status control function by reusing the camera and adding an image processing software to perform different functions. Specifically, the following heterogeneous redundancies can be implemented reusing video surveillance camera [#13] with the necessary processing software and communication interfaces: door-open detection [#7], door-closed detection [#8], door velocity [#9], and door-obstacle detection [#10].

As a result of the compatibility analysis, the designer can select different homogeneous or heterogeneous redundancy strategies for each subfunction. Apart from the identified heterogeneous redundancies, it is possible to add homogeneous redundancies duplicating existing sensors. For instance, for the door status control function in Table 2 the homogeneous and heterogeneous redundancy decisions in Table 3 can be adopted. Communication integrates MVB and Ethernet communication networks and their connecting gateway.

Table 3: Redundancy strategies for door status control main function.

Implementation	**Subfunction**			
Type	***Door Open Detection***	***Door Closed Detection***	***Obstacle Detection***	***Door Velocity***
Nominal	PU_{Door}, OpenSensor	PU_{Door}, ClosedSensor	PU_{Door}, ObstacleSensor	PU_{Door}, VelocitySensor
Heterogeneous	Camera, PU_{Cam}, $SW_{OpenDet}$, communication	Camera, PU_{Cam}, $SW_{CloseDet}$, communication	Camera, PU_{Cam}, $SW_{ObstDet}$, communication	Camera, PU_{Cam}, SW_{Speed}, communication
Homogeneous	PU_{Door}, OpenSensor2	PU_{Door}, ClosedSensor2	PU_{Door}, ObstacleSensor2	PU_{Door}, VelocitySensor2

There are several approaches in the diagnostics and fault-tolerant control community focused on identifying analytic redundancies systematically[18]. A number of approaches in this area evaluate if it is possible to provide the same service with a combination of remaining sensors, i.e., if there exists an alternative analytic equation, which uses a different set of variables (resources) to provide the same service. The identification of redundancies focuses on the relations among system equations, and variables. That is, if there exists redundant information about the system structure (i.e., if there are more equations than variables to be determined) there may also exist alternative ways to define a variable.

The exhaustive characterization and mathematical formulation of complex systems is not trivial and in some cases is infeasible. The identification of analytic redundancies is typically feasible at subsystem level, but the complexity of the mathematical formulation increases dramatically at system level. Additional complexity exists in highly networked scenarios where systems consists of many subsystems, which are all interconnected through a communication network. In general, the formal identification and categorisation of heterogeneous redundancies for complex systems is a challenging task. This is pronounced in the case of non-evident redundancies raised from forced compatibilities because there is no direct relationship between them.

Reconfiguration strategies integrate the functional model with redundancies. They define all possible realizations of the main function comprised of the necessary subfunctions and prioritized implementations. The prioritization is based on the weighted sum of functional degradation, failure probability and cost of the implementation[4]. The functional degradation depends on the relative physical distance (applicable for heterogeneous redundancies arising from natural compatibilities). For heterogeneous redundancies raising from forced compatibilities, the designer's knowledge is necessary.

5.3 Extended Functional Modelling Approach

The Extended Functional Modelling Approach augments the functional model by adding health management functions and implementations: fault detection to detect the incorrect operation of an implementation and reconfiguration to recover from implementation failures. We have defined the following mechanisms and protocols for fault detection and reconfiguration subfunctions:

- Fault detection (FD): each subfunction has an associated fault detection subfunction (FD_SF). The FD_SF is located at the destination processing unit where the information of the source processing unit is used to detect communication omission failures directly.
- Reconfiguration (R): each subfunction has its own reconfiguration subfunction (R_SF), which receives fault detection (FD_SF) signals and sends reconfiguration signals to subfunction implementations.
- Fault detection of the reconfiguration (FD_R): each reconfiguration implementation (R_SF) has its own fault detection mechanism (FD_R_SF) implemented in keepalive configuration. Each R_SF implementation sends keepalive signals to all their FD_R_SF implementations to indicate that it is operating. In the absence of a keepalive signal during a time-slot, an R_SF implementation is assumed to have failed. When this happens, the FD_R_SF implementation sends an activation signal to the available R_SF implementation with the highest priority.
- Communication is considered at resource level.

There does not exist a uniquely valid solution when allocating health management implementations. The adopted decisions predefine the behaviour of health management mechanisms so that it is possible to design and evaluate HW/SW architectures systematically.

Since fault detection and reconfiguration are subfunctions of a given main function, they are also modelled using tokens (FD_SF, R_SF, FD_R_SF). Accordingly it is possible to analyse alternative fault detection and reconfiguration strategies. Figure 5 describes the closed-loop operation of a system deployed in a highly networked scenario including input, control and output subfunctions. The operation of the HW/SW architecture is described for the output subfunction with redundancies. Overlapped rectangles describe alternative implementations for the same subfunction.

Extending the functional model of the door status control main function in Table 2, Table 4 displays the HW/SW architecture including the identified heterogeneous redundancies (cf. Table 3) and their health management mechanisms. Namely, for each subfunction with redundancies: a single fault detection implementation (FD_SF), duplicated reconfiguration implementations (R_SF), and duplicated fault detection of the reconfiguration (FD_R_SF) implementations have been selected.

The HW/SW architecture design step can be automated[4] and implemented in real systems[5]. As for the automation, the token-based annotations make it possible to parse the HW/SW architecture from a design model (e.g. Simulink[37]) which includes designers decisions with respect to the level and type of redundancy and health management strategies. For implementation, each processing unit needs a

Table 4: HW/SW architecture of the door status control main function.

MF	*PL*	*SF*	*Resources*	*MF*	*PL*	*SF*	*Resources*
DSC	Train. Car_1. $Zone_A$. Door	EDD	PU_{TCMS1}, SW_{TCMS1}	DSC	Train. Car_1. $Zone_A$. Door	FD_R_DCD	PU_{Door}, $SW_{FD\_R\_DCD}$, Comm
		EDD	PU_{TCMS2}, SW_{TCMS2}			FD_R_DCD	PU_{Cam}, $SW_{FD\_R\_DCD}$, Comm
		EDP	EDD, PU_{Driver}, EnableBut., Comm			OD	PU_{Door}, ObstacleSensor
		DCC	PU_{Driver}, $CloseButton_{Driver}$			OD	Camera, PU_{Cam}, $SW_{ObstacleDet}$, Comm
		DOC	PU_{Driver}, $OpenButton_{Driver}$			FD_OD	PU_{Door}, SW_{FD_OD}, Comm
		DOC	PU_{Cam}, $OpenButton_{Passenger}$			R_OD	PU_{Door}, SW_{R_OD}
		DOD	PU_{Door}, OpenSensor			R_OD	PU_{Cam}, SW_{R_OD}, Comm
		DOD	Camera, PU_{Cam}, $SW_{OpenDet}$, Comm			FD_R_OD	PU_{Door}, $SW_{FD\_R\_OD}$, Comm
		FD_DOD	PU_{Door}, SW_{FD_DOD}, Comm			FD_R_OD	PU_{Cam}, $SW_{FD\_R\_OD}$, Comm
		R_DOD	PU_{Door}, SW_{R_DOD}			DV	PU_{Door}, SpeedSensor
		R_DOD	PU_{Cam}, SW_{R_DOD}, Comm			DV	Camera, PU_{Cam}, $SW_{DoorVelocity}$, Comm
		FD_R_DOD	PU_{Door}, $SW_{FD\_R\_DOD}$, Comm			FD_DV	PU_{Door}, SW_{FD_DV}, Comm
		FD_R_DOD	PU_{Cam}, $SW_{FD\_R\_DOD}$, Comm			R_DV	PU_{Door}, SW_{R_DV}
		DCD	PU_{Door}, ClosedSensor			R_DV	PU_{Cam}, SW_{R_DV}, Comm
		DCD	Camera, PU_{Cam}, $SW_{CloseDet}$, Comm			FD_R_DV	PU_{Door}, $SW_{FD\_R\_DV}$, Comm
		FD_DCD	PU_{Door}, SW_{FD_DCD}, Comm			FD_R_DV	PU_{Cam}, $SW_{FD\_R\_DV}$, Comm
		R_DCD	PU_{Door}, SW_{R_DCD}			DCA	EDP, DCC, DOC, DOD, DCD, DV, OD, PU_{Door}, SW_{DSC}, Comm
		R_DCD	PU_{Cam}, SW_{R_DCD}, Comm			DM	DCA, PU_{Door}, Motor

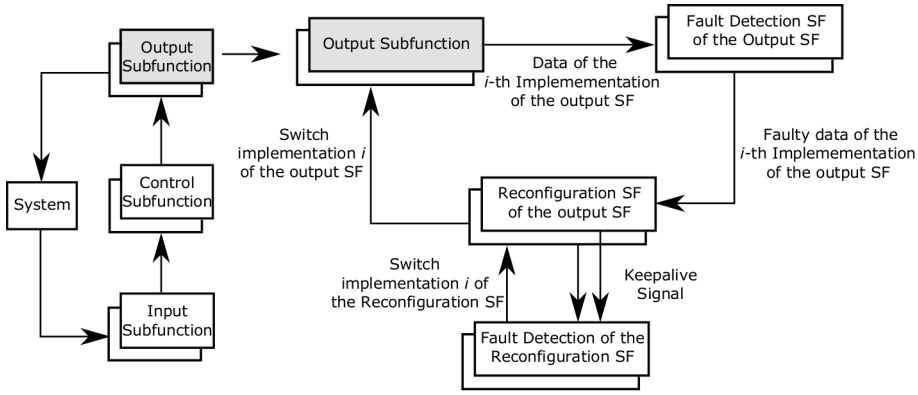

Figure 5: Operation of the HW/SW architecture.

wrapper that ensures the interchangeability between compatible implementations and a reconfiguration mechanism to redirect its information. Furthermore, the units with FD_R_SF implementations require monitoring keepalive signals to control the correct operation of the active R_SF implementation[5].

6 Dependability and Cost Evaluation Approach for Repairable Systems

6.1 Concepts and Notation

The failure model of the HW/SW architectures considers the possible failure modes of its health management mechanism and functional implementations: fault detection implementations (FD_SF, FD_R_SF) fail in omission (O) when they do not detect an occurred failure, and in false positive (FP) when they falsely report a failure that has not occurred; reconfiguration implementations fail in omission when they fail to act on needed reconfiguration; and failure of subfunction implementations (SF) cover omission and incorrect value failure modes.

All possible failures of all system subfunction implementations (SF, FD_SF, R_SF, FD_R_SF) are defined at the implementation-level (i.e., [MF].[PL].[SF].[Impl] *Failure*) with respect to failures of the implementation resources. Based on the combination of implementation-level failures, subfunction-level failures are defined systematically ([MF].[PL].[SF] *Failure*).

Implementations are reconfigured sequentially for non-repairable systems[5]. However, for repairable systems, it is necessary to check the status of all subfunction implementations to know which implementation is active and reconfigure the implementation with the highest priority (cf. Figure 1). Implementation *i* becomes active if at initialisation it has the highest priority among the implementations for the same subfunction, or when the active implementation fails and implementation *i* has the highest priority among the available implementations.

The logical and temporal combination of failure and repair events are specified using repairable Dynamic Fault Tree gates (cf. Table 5).

Table 5: Repairable Dynamic Fault Tree gates.

Gate Notation	**Gate Behaviour**
Y=**AND**(A,B)	If A fails and B fails, then Y fails. If A or B is repaired, then Y is also repaired.
Y=**OR**(A,B)	If A fails or B fails, then Y fails. If A or B is repaired, then Y is also repaired.
Y=**PAND**(A,B)	If A fails before the failure of B or at the same time, then Y fails. If A is repaired, then Y is repaired and another sequence of A failing before B is needed to cause the failure of Y.

The use of these gates is limited to expressing certain events with predefined failure and repair logic, but more flexible failure and repair specification logics are also needed to model non-predefined random events (see Subsection 6.3).

Table 6 defines the notations of the failure events and working events according to their subfunction and failure modes. For brevity, in subsequent characterizations we omit the common part ([MF].[PL]).

Table 6: Notation of failure and working events.

Notation	***Failure Logic***	***Notation***	***Failure/Working Logic***
$\mathcal{F}_{\mathrm{X}}$	X failure	$\mathcal{W}_{\mathrm{X}}$	X working
$\mathcal{F}_{\mathrm{SF}}$	[SF] failure	$\mathcal{W}_{\mathrm{SF}_i}$	[SF].[Impl$_i$] working = **NOT**($\mathcal{F}_{\mathrm{SF}_i}$)
$\mathcal{F}_{\mathrm{SF}_i}$	[SF].[Impl$_i$] failure	$\mathcal{F}_{\mathrm{R}}$	[R_SF] failure
$\mathcal{F}_{\mathrm{FD}}$	[FD_SF] failure	$\mathcal{F}_{\mathrm{R}_i\ \mathrm{O}}$	[R_SF].[Impl$_i$] omission
$\mathcal{F}_{\mathrm{FD\ FP}}$	[FD_SF] false positive	$\mathcal{F}_{\mathrm{FD\_R}_i\ \mathrm{FP}}$	[FD_{[R_SF].[Impl$_i$]}] false positive
$\mathcal{F}_{\mathrm{FD}_i}$	[FD_SF].[Impl$_i$] failure	$\mathcal{F}_{\mathrm{FD\_R}_i\ \mathrm{O}}$	[FD_{[R_SF].[Impl$_i$]}] omission
$\mathcal{F}_{\mathrm{FD}_i\ \mathrm{O}}$	[FD_SF].[Impl$_i$] omission	$\mathcal{F}_{\mathrm{R}_i\ \mathrm{O/FP}}$	[R_SF].[Impl$_i$] omission or FP = **OR**($\mathcal{F}_{\mathrm{R}_i\ \mathrm{O}}$, $\mathcal{F}_{\mathrm{FD\_R}_i\ \mathrm{FP}}$)
$\mathcal{F}_{\mathrm{SF}_i\ \mathrm{FP}}$	[SF].[Impl$_i$] failure or FP = **OR**($\mathcal{F}_{\mathrm{SF}_i}$, $\mathcal{F}_{\mathrm{FD\ FP}}$)	$\mathcal{F}_{\mathrm{SF}_i\ \mathrm{FP\ |\ Act}}$	[SF].[Impl$_i$] fail or FP while active = **OR**($\mathcal{F}_{\mathrm{SF}_i\ |\ \mathrm{Act}}$, $\mathcal{F}_{\mathrm{FD\ FP}}$)
$\mathcal{F}_{\mathrm{SF}_i\ |\ \mathrm{Act}}$	[SF].[Impl$_i$] fail while active	$\mathcal{F}_{\mathrm{FD}_i\ \mathrm{O}\ |\ \mathrm{Act}}$	[FD_SF].[Impl$_i$] omission while active
$\mathcal{F}_{\mathrm{SF\_Dest}_i\ |\ \mathrm{Act}}$	[SF_Dest].[Impl$_i$] fail while active		

The failure specification of each resource is defined by sampling randomly the failure and repair times according to their cumulative distribution functions along the system lifetime. The methodology supports any cumulative distribution function, but for the sake of demonstration and without loss of generality, in subsequent probabilistic characterizations exponential failure distributions are assumed. In line with this assumption, the failure specification of resources ($\mathcal{F}_{\mathrm{Res}}$) is defined according to their failure rates (λ_{Res}) and repair rates (μ_{Res}). $\mathcal{F}_{\mathrm{Res}}$ can be seen as continuous-time Markov chains with working and failed states, where the transitions between these states are determined by λ_{Res} and μ_{Res} parameters.

The failure specification of a subfunction's i-th implementation ([SF].[Imp$_i$] *Failure*) comprised of N resources is defined as follows:

$$\mathcal{F}_{\mathrm{SF}_i} = \mathbf{OR}(\mathcal{F}_{\mathrm{Res}_1},\ \mathcal{F}_{\mathrm{Res}_2},\ \ldots,\ \mathcal{F}_{\mathrm{Res}_N}) \tag{2}$$

The same equation holds for the specification of the omission failures of: fault detection (FD_SF — $\mathcal{F}_{\mathrm{FD}_i\ \mathrm{O}}$), reconfiguration (R_SF — $\mathcal{F}_{\mathrm{R}_i\ \mathrm{O}}$), and fault detection of the reconfiguration (FD_R_SF — $\mathcal{F}_{\mathrm{FD\_R_i\ O}}$). Accordingly, false positive failures of fault detection implementations ($\mathcal{F}_{\mathrm{FD\ FP}}$ and $\mathcal{F}_{\mathrm{FD\_R_i\ FP}}$) are specified with failure and repair distributions and parameters.

6.2 Dependability Analysis Algorithm

The dependability analysis algorithm defines compositionally combinations of subfunction implementation failures that prevent the HW/SW architecture from performing its intended subfunction. The failure of any subfunction necessary for a main function provokes the immediate failure of a main function. Hence, from this point onwards, we will only consider the failure of a subfunction. To express these events we use equations with the logic gates defined in Table 5.

The subfunction fails ($\mathcal{F}_{\mathrm{SF}}$) when all implementations have failed ($\mathcal{F}_{\mathrm{All\ Impl.}}$), an implementation fails and reconfiguration does not happen (failure unresolved, $\mathcal{F}_{\mathrm{Unresolved}}$), or its input dependencies have failed ($\mathcal{F}_{\mathrm{Dependencies}}$):

$$\mathcal{F}_{\mathrm{SF}} = \mathbf{OR}(\mathcal{F}_{\mathrm{All\ Impl.}},\ \mathcal{F}_{\mathrm{Unresolved}},\ \mathcal{F}_{\mathrm{Dependencies}}) \tag{3}$$

Assuming that we have N_{SF} implementations of the subfunction, the $\mathcal{F}_{\mathrm{All\ Impl.}}$ event happens when each implementation fails or is detected as failed:

$$\mathcal{F}_{\mathrm{All\ Impl.}} = \mathbf{AND}(\mathcal{F}_{\mathrm{SF_1\ FP}},\ \ldots,\ \mathcal{F}_{\mathrm{SF}_{N_{SF}}\ \mathrm{FP}}) \tag{4}$$

The failure unresolved ($\mathcal{F}_{\mathrm{Unresolved}}$) occurs when the active implementation fails and either the fault is not detected (failure undetected event) or the reconfiguration itself fails (reconfiguration failed event). For each implementation there are different failure unresolved events ($\mathcal{F}_{\mathrm{Unr.\ Imp}_i}$) because each implementation has different failure probabilities:

$$\mathcal{F}_{\mathrm{Unresolved}} = \mathbf{OR}(\mathcal{F}_{\mathrm{Unr.\ Imp_1}}, \ldots,\ \mathcal{F}_{\mathrm{Unr.\ Imp}_{N_{SF}}}) \tag{5}$$

To define the failure unresolved event ($\mathcal{F}_{\mathrm{Unr.\ Imp}_i}$) we introduce two new events. The first event occurs when first the reconfiguration subfunction fails and then the i^{th} implementation of the subfunction fails when it is active (reconfiguration sequence failure, $\mathcal{F}_{\mathrm{R\ Seq.}_i}$):

$$\mathcal{F}_{R\ Seq._i} = \mathbf{PAND}(\mathcal{F}_{\mathrm{R}},\ \mathcal{F}_{\mathrm{SF}_i\ \mathrm{FP}\ |\ \mathrm{Act}}) \tag{6}$$

The second event occurs when first the fault detection of the subfunction fails and then the i^{th} implementation of the subfunction fails when it is active (fault detection sequence failure, $\mathcal{F}_{\mathrm{FD\ Seq.}_i}$):

$$\mathcal{F}_{\mathrm{FD\ Seq.}_i} = \mathbf{PAND}(\mathcal{F}_{\mathrm{FD}},\ \mathcal{F}_{\mathrm{SF}_i\ |\ \mathrm{Act}}) \tag{7}$$

Accordingly, the failure unresolved event of the i^{th} implementation ($\mathcal{F}_{\text{Unr. Imp}_i}$) occurs when either the fault detection sequence ($\mathcal{F}_{\text{FD Seq.}_i}$) fails or the reconfiguration sequence ($\mathcal{F}_{\text{R Seq.}_i}$) fails:

$$\mathcal{F}_{\text{Unr. Imp}_i} = \mathbf{OR}(\mathcal{F}_{\text{FD Seq.}_i},\ \mathcal{F}_{\text{R Seq.}_i}) \tag{8}$$

Dependencies address the influence of Input (I) and Control (C) subfunctions to influence on Control and Output (O) subfunctions respectively. A Control subfunction failure impacts directly the output subfunction failure (C→O). The influence of an input subfunction on a control subfunction depends on the control configuration of the system, i.e. whether this is Closed Loop (C_CL) or Open Loop (C_OL):

$$\mathcal{F}_{\text{Dependencies}} = \mathbf{OR}(\mathcal{F}_{\text{Dep. C\_CL}},\ \mathcal{F}_{\text{Dep. C\_OL}}) \tag{9}$$

Assuming that $\mathcal{W}_{\text{C\_X}}$=**OR**($\mathcal{W}_{\text{C\_X}_1}, \ldots, \mathcal{W}_{\text{C\_X}_{N_\mathcal{W}}}$) means that any of the $N_\mathcal{W}$ implementations of the C_X subfunction are working (where $X = \{CL, OL\}$), equations in (10) describe the different input subfunctions that affect each control configuration (I_CL→C_CL, I_OL→C_OL). $\mathcal{F}_{\text{Dep. C\_OL}}$ may not happen because the open loop control generally does not have input dependencies:

$$\begin{aligned} \mathcal{F}_{\text{Dep. C\_CL}} &= \mathbf{AND}(\mathcal{W}_{\text{C\_CL}},\ \mathcal{F}_{\text{I\_CL}}) \\ \mathcal{F}_{\text{Dep. C\_OL}} &= \mathbf{AND}(\mathcal{W}_{\text{C\_OL}},\ \mathcal{F}_{\text{I\_OL}}) \end{aligned} \tag{10}$$

The reconfiguration failure is a special subfunction and therefore $\mathcal{F}_{\text{R}}$ is developed like Eq. (3), except that there are no additional dependencies:

$$\mathcal{F}_{\text{R}} = \mathbf{OR}(\mathcal{F}_{\text{All R Impl.}},\ \mathcal{F}_{\text{R Unresolved}}) \tag{11}$$

$\mathcal{F}_{\text{All R Impl.}}$ indicates the failure of all reconfiguration implementations and $\mathcal{F}_{\text{R Unresolved}}$ designates the failure unresolved condition of the reconfiguration. Assuming *M* reconfiguration implementations:

$$\mathcal{F}_{\text{All R Impl.}} = \mathbf{AND}(\mathcal{F}_{\text{R}_1\text{ O/FP}},\ \ldots,\ \mathcal{F}_{\text{R}_M\text{ O/FP}}) \tag{12}$$

$\mathcal{F}_{\text{R Unresolved}}$ happens when *M* implementations of the reconfiguration's fault detection fail simultaneously and it is a direct consequence of design choice: all fault detection implementations of the reconfiguration (FD_R_SF) are active and homogeneous redundancies (keepalive implementations):

$$\mathcal{F}_{\text{R Unresolved}} = \mathbf{AND}(\mathcal{F}_{\text{FD\_R}_1},\ \ldots,\ \mathcal{F}_{\text{FD\_R}_M}) \tag{13}$$

The false positive of the reconfiguration's fault detection occurs when all reconfiguration's fault detection implementations raise the false positive condition simultaneously. Although the system may operate correctly when a false positive occurs, it has to assume that the information provided by the fault detection is correct, since there is no mechanism to detect the incorrect operation of fault detection. The fault detection failure $\mathcal{F}_{\text{FD}}$ depends on the operation of the destination subfunction (SF_{Dest}), because the fault detection implementation is located at the same processing unit. Hence, $\mathcal{F}_{\text{SF\_Dest}}$ influences directly $\mathcal{F}_{\text{FD}}$.

When the fault detection implementation fails, the change of destination subfunction's (SF_{Dest}) implementation determines its reconfiguration. We assume that the change of destination subfunction's implementation activates the corresponding fault detection implementation and the previous one is deactivated. Eq. (14) describes the fault detection subfunction failure case when fault detection subfunction has K implementations:

$$\mathcal{F}_{\text{FD}} = \mathbf{OR}(\mathcal{F}_{\text{FD\_Dest}_1 \mid \text{Act}}, \ldots, \mathcal{F}_{\text{FD\_Dest}_K \mid \text{Act}}) \tag{14}$$

The failure of the i^{th} fault detection implementation while it is active ($\mathcal{F}_{\text{FD\_Dest}_i \mid \text{Act}}$) expresses the next event: either the i^{th} destination subfunction or the i^{th} fault detection implementation fail while active (note that i^{th} fault detection and SF_Dest_i implementation are located at the same processing unit):

$$\mathcal{F}_{\text{FD\_Dest}_i \mid \text{Act}} = \mathbf{OR}(\mathcal{F}_{\text{SF\_Dest}_i \mid \text{Act}}, \mathcal{F}_{\text{FD}_i \text{ O} \mid \text{Act}}) \tag{15}$$

To avoid creating loops, the influence of dependencies is taken into account at the subfunction's failure level (cf. Eq. (3)). At this level, the failure of any dependent subfunction leads directly to the subfunction failure.

6.3 Implementation

Stochastic Activity Networks (SAN)[38] meet all the requirements to specify the dependability evaluation model of HW/SW architectures including the specification of: time-dependant scenarios; modular system behaviour; repair behaviour; any cumulative distribution function; and user-defined reconfiguration strategies (cf. Section 2).

6.3.1 Preliminaries on SAN

SAN was first introduced in the mid-1980s[39] and it has been used for performance, dependability and performability evaluations[6,40,41]. SAN makes use of reduced base models[42] so as to alleviate the state-explosion problem and it extends stochastic Petri Nets generalizing the stochastic relationships and adding mechanisms for hierarchical models[38]. Figure 6 shows the SAN modelling constructs.

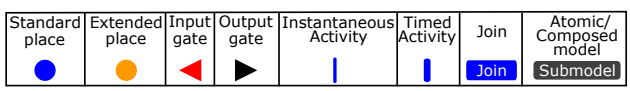

Figure 6: SAN modelling constructs.

Places represent the state of the modelled system. Each place contains tokens defining the marking of the place: a standard place contains an integer number of tokens, while extended places contain data types other than integers (e.g. float, array). We will denote the marking function of the place x as $m(x)$, e.g. $m(x) = 1$ means that the place x has a marking equal to one.

There are two types of activities: instantaneous which complete in negligible amount of time; and timed whose duration has an effect on the system performance and their completion time can be a constant or a random value. The random value is ruled by a probability distribution function defining the time to fire the activity.

Activities fire based on the conditions defined over the marking of the network and their effect is to modify the marking of the places. The completion of an activity of any kind is enabled by a particular marking of a set of places. The presence of at least one token in each input place enables the firing of the activity removing the token from its input place(s) and placing it in the output place(s).

Another way of enabling activities consists of utilising input and output gates. Gates make SAN general and powerful enough to model complex real situations. They determine the marking of the network via employing user-defined C++ rules. Input gates control the enabling of activities and define the marking changes that will occur when an activity completes. A set of places is connected to the input gate and the input gate is connected to an activity. A Boolean condition enables the activity connected to the gate and a function determines the effect of the activity completion on the marking of the places connected to the gate. Output gates specify the effect of activity completion on the marking of the places connected to the output gate. An output function defines the marking changes that occur when the activity completes.

SAN models which include the specified SAN elements form a SAN atomic model (see Figure 10 "Reusable Block" column). The join operator links SAN models through a compositional tree structure in a unique composed model (e.g., see Figure 8). It is possible to link atomic models, composed models, or combinations thereof. Composed and atomic SAN models are linked through join operators using shared places between them. Thus, the analyst can focus on specific characteristics through fit-for-purpose atomic/composed models and later join independently validated models to obtain a more complex composed model.

The performance measurements are carried out through reward functions defined over the designed model. Reward functions are defined based on the marking of the network (state reward function) or completion of activities (impulse reward function) and they are evaluated as the expected value of the reward function. For a complete and formal definition of SAN please refer to[38].

6.3.2 Dependability Evaluation Approach Specification in SAN

Figure 7 shows the specification of the dependability analysis algorithm comprised of the following models and activities:

- **Functional Modelling**: for each subfunction (SF) its resources, implementations, and the reconfiguration logic are specified using SAN atomic models. The same modelling process applies for each fault detection (FD_SF), reconfiguration (R_SF) and reconfiguration's fault detection (FD_R_SF) subfunction implementations.

- **Failure Logic Modelling**: the failure logic of the gates used in Eqs. (2)-(15) are modelled in SAN.

- **SAN Synthesis**: according to the dependability analysis algorithm, SAN composed models are created linking resources, implementations, reconfiguration logic and failure logic. Composed models are constructed by creating shared places between implementations and failure gates. They define implementation-level failures (cf. Eq. (2)) and they are linked to define subfunction and main function level failures (cf. Eq. (3)).

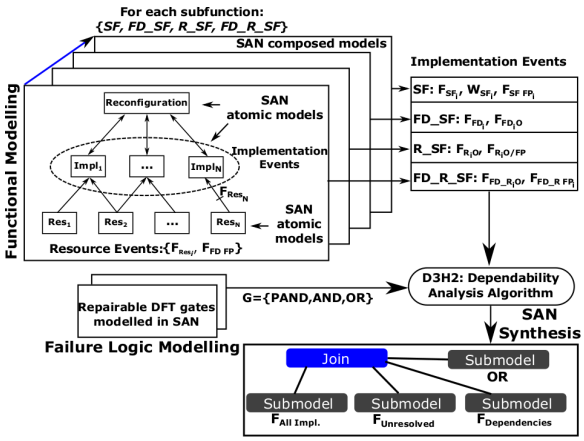

Figure 7: Dependability Evaluation Approach specification process in SAN.

Functional Modelling: for each subfunction its different implementations, resources, and reconfiguration logic are specified using SAN atomic models. For instance, assuming that the implementation $Impl_1$ is comprised of resources Res_1 and Res_2, Figure 8 shows the SAN atomic specification of (a) resources (Res_1); (b) implementations ($Impl_1$); and (c) the SAN composed model that links implementations and resources via shared places.

As modelled in the resource specification (Figure 8a), Res_1 (and Res_2) transits between working and failed states according to its failure and repair cumulative distribution functions ($F(t)$, $R(t)$). Initially resources are assumed to be operative (<m(Res$_1$ Working), m(Res$_1$ Failure)> = <1, 0>) and implementations can be in working or standby state, e.g., $Impl_1$ is working (<m(Impl$_1$ Working), m(Impl$_1$ Failure), m(Impl$_1$ Standby)> = <1, 0, 0>).

According to the atomic implementation specification, when Res_1 or Res_2 fails, $Impl_1$ switches to failure state (see the logic in F_Impl$_1$ input gate). When both resources Res_1 and Res_2 are repaired, $Impl_1$ switches to standby state (see the logic in R_Impl$_1$ input gate). If $Impl_1$ is in standby state and receives a reconfiguration signal (m($Impl_1$ Reconfigure)=1), then instantaneously returns to the working state (see atomic model of the implementation specification — Figure 8b).

The composed model of the implementation links atomic models of resources and implementations sharing their dependent places: Res$_1$ Failure and Res$_2$ Failure (Figure 8c). This modelling process is repeated for all the implementations and their constituent resources.

After specifying all the implementations and resources, it is necessary to define the reconfiguration logic between implementations. Figure 9 shows the reconfiguration process for $Impl_1$ and $Impl_2$

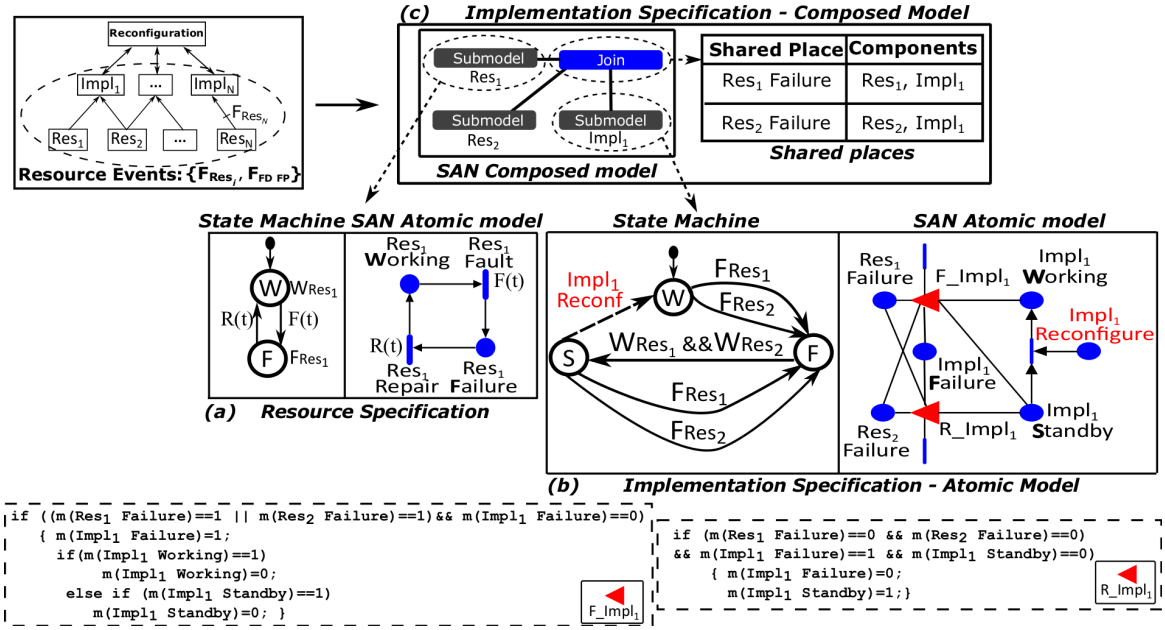

Figure 8: Specification of implementations and resources.

assuming that $Impl_1$ has higher priority than $Impl_2$. The SAN atomic model of the reconfiguration (`Reconfig_SF`) defines the reconfiguration process:

- The implementation with the highest priority starts operating ($Impl_1$).
- When $Impl_1$ fails the next implementation in standby state with the highest priority is activated ($Impl_2$).
- When the failed implementation is repaired, it returns to the standby state and it remains in standby state until the implementation that is active fails.
- When the implementation which does not have the highest priority fails, standby implementations are checked according to their priority. In this case, if $Impl_1$ is in standby state when $Impl_2$ fails, it returns to the active operation.

This process is extendible to N implementations and the implementation reconfiguration priorities are determined according to *if-else-if* statements and implementations states.

To implement the reconfiguration logic the atomic model `Reconfig_SF` in Figure 9 is joined with the composed models of $Impl_1$ and $Impl_2$ (cf. Figure 8c) creating shared places between the implementations and the reconfiguration logic for each implementation: $\texttt{Impl}_i$ `Failure`, $\texttt{Impl}_i$ `Reconfigure`, and $\texttt{Impl}_i$ `Standby`, where i identifies the implementation, i={1, 2}.

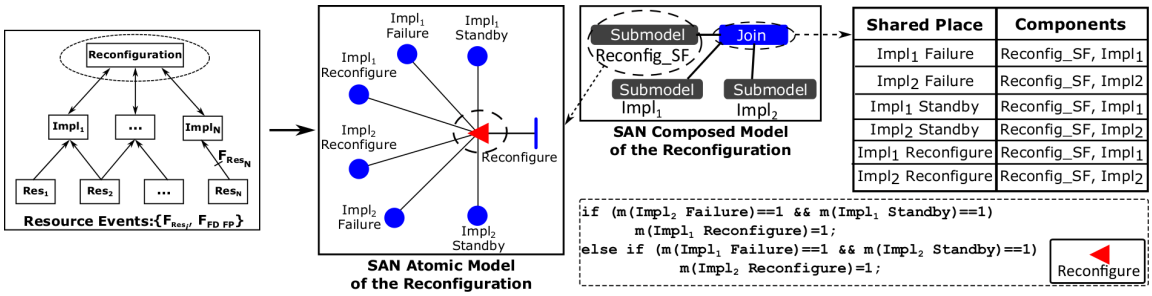

Figure 9: Specification of the reconfiguration process.

Failure Logic Modelling: in order to implement the logic in the equations of the dependability analysis algorithm it is necessary to model in SAN the logic of repairable Dynamic Fault Tree gates — see Table 5. Figure 10 shows the specification of repairable Dynamic Fault Tree gates in SAN using state machines and their corresponding SAN model. In the state machine the initial state is indicated with an arc, failure states are identified with doubled circles, and F_x and R_x indicate failure and repair events of x. The resultant reusable blocks are used to create the equations of the dependability analysis algorithm systematically.

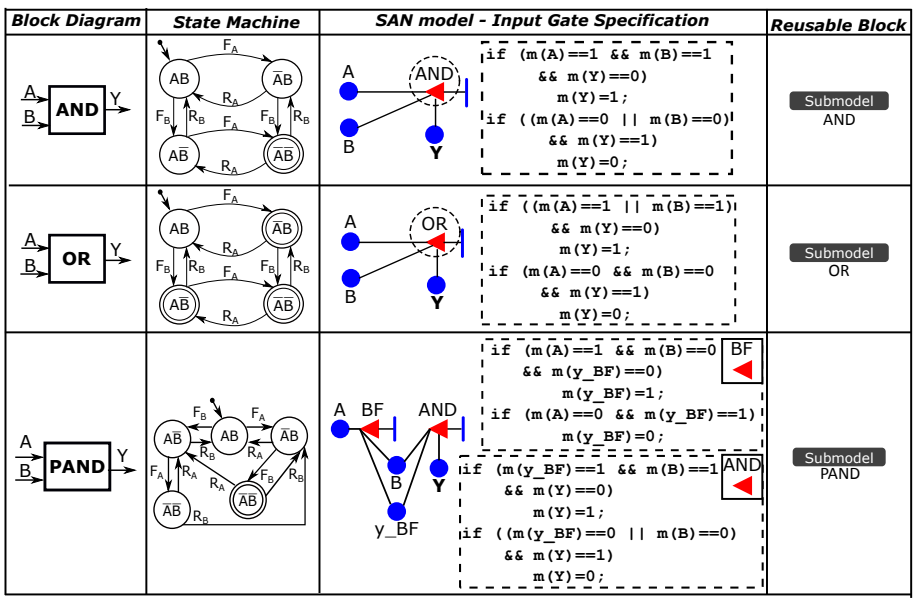

Figure 10: Specification of repairable Dynamic Fault Tree gates in SAN.

Note that the repairable Dynamic Fault Tree gates in Figure 10 are directly extendible to gates with N inputs and they can be used in a broader context for the evaluation of any complex repairable Dynamic Fault Tree model. The behaviour of the repairable gates have been validated using other repairable Dynamic Fault Tree analysis tools[24].

SAN Synthesis: linking the design and operation logic for all the system resources, implementations, and subfunctions and then connecting them with failure gates leads to synthesis in SAN of the equations of the dependability analysis algorithm. The algorithm is applied bottom-up using Eqs. (2)-(15), starting from resources and implementations (Eq. (2)) up to the subfunction failure (Eq. (3)).

For instance, Figure 11 shows $\mathcal{F}_{\text{All Impl.}}$ event (cf. Eq. (4)) assuming that the subfunction under study is comprised of two implementations.

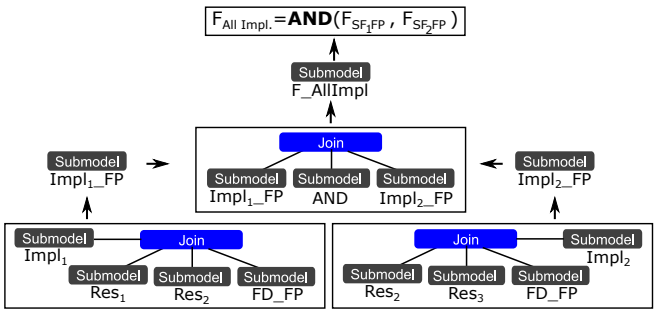

Figure 11: SAN synthesis example for the $\mathcal{F}_{\text{All Impl.}}$ event defined in Eq. (4).

The same modelling process applies to the remainder of the equations of the Dependability Evaluation Approach. In this way compositional dependability evaluation of complex reconfigurable systems is achieved by linking the dependability analysis algorithm with component-based SAN models of system elements. Note that the reconfiguration model for each subfunction (cf. Figure 9) is linked at the subfunction failure level (cf. Eq. (3)) so as to reconfigure subfunction implementations consistently.

6.4 Door Status Control Case Study Application

Starting from the functional model of the door status control in Table 2, we have identified heterogeneous redundancies for different subfunctions. For instance, it is possible to reuse a video surveillance camera to provide redundancies for door open detection, door closed detection, obstacle detection; and door velocity subfunctions — see Subsection 5.2. Table 3 displays alternative redundancy strategies that can be considered at the design phase.

To use these redundancies, the HW/SW architecture is designed adding fault detection and reconfiguration mechanisms. In the HW/SW architecture displayed in Table 4 we have assumed that for each subfunction with redundancies we have one fault detection subfunction (FD_SF), two reconfiguration (R_SF), and two fault detection of the reconfiguration (FD_R_SF) implementations.

The cost assessment of the designed architecture is carried out by adding up the cost of hardware and software resources. The cost of software components is quantified by considering their development cost assuming that it will be paid off in X years (let us assume X=4 years for calculation purposes). We classify four types of SW components: fault detection (SW_FD), reconfiguration (SW_R), fault detection of the reconfiguration (SW_FD_R) and Control-Detector (SW_Det). The development costs for

each of these four software components is considered once for different subfunction implementations: once developed, they are adapted for the related subfunction implementations.

This assumption is adopted because the grouped subfunction implementations are closely related and they do not need a significant development cost (the cost of N variants is not N times the cost of a single software variant[43]): fault detection implementations adapt to different subfunctions modifying subfunction-specific time/value thresholds. The cost of development of reconfiguration implementations does not differ for different subfunctions because the reactivation logic remain. The fault detection implementations of a reconfiguration differ only in the keepalive timeout and the development is independent of any subfunction. All the control-detector software implementations have a similar logic.

Hardware cost is evaluated using the sensors, controllers and actuator costs obtained from suppliers. The labour cost related with mounting/testing is considered for sensors and actuators assuming 10 minutes per sensor (actuator) at a rate of 60 €/hour. Downtime cost is measured as the combination of travels lost while the train was stopped (*travels_lost*); people in each travel (*people_travel*); and cost of a ticket per person (*ticket_cost*):

$$downtime\_cost = travels\_lost \times people\_travel \times ticket\_cost$$

$$travels\_lost = \frac{travels}{hour} \times downtime$$

$$downtime = failure\ probability \times mission\ time$$

We assume that we do not have to stop the whole train to fix a failure in a car. Besides, we adopt the following values for a short-distance train ($\leq$ 50 km): $\frac{travels}{hour}$ = 2; *people_travel* = 20; *ticket_cost* = 1 €; *mission time* = 30 years. We will evaluate the *failure probability* at T = 30 years time instant.

Regarding their failure rate values, resources with the same characteristics have been grouped in Table 7: pressure sensor covers open, closed and obstacle detection sensors; PU gathers characteristics of all different processing units; and communications include MVB and Ethernet communication protocols and their gateway. Regarding software components, plausible values are assumed. The repair rate for all components is assumed to be μ = 0.5 y^{-1}.

Table 7: Failure rate & cost values.

Resource	λ (yr^{-1})	*Cost (€)*
SW_Det, SW_HM	1E-2	80 each
SW_FP	1E-2	-
Pressure Sensor[44] + Mounting	1.6E-2	20 + 60€/hr
Speed Sensor[44] + Mounting	1.8E-2	20 + 60€/hr
Camera[45]	9.43E-2	-
PU[46]	3.87E-2	30
Communications	5E-3	200

Table 8: Analysed redundancy strategies.

ID	*Configuration*
#1	4 heterogeneous redundancies (cf. Table 4)
#2	3 heterogeneous redundancies: DCD, DOD, DV; 1 homogeneous redundancy: OD
#3	2 heterogeneous redundancies: DCD, DOD; 2 homogeneous redundancies: OD, DV
#4	1 heterogenous redundancy: DCD; 3 homogeneous redundancies: OD, DV, DOD
#5	4 homogeneous redundancies

We have analysed the failure probabilities of different HW/SW architectures with alternative redundancy strategies by applying the dependability analysis algorithm (cf. Subsection 6.2) and synthesizing

the equations of Dependability Evaluation Approach in SAN (cf. Subsection 6.3.2). Table 8 displays analysed redundancy strategies using the redundancies displayed in Table 3 and Table 9 displays the implementations of the health management mechanisms used for the set of subfunctions with redundancies denoted as SF={DOD, DCD, OD, DV}.

Table 9: Health management implementations and resources.

Implementation	*FD_SF*	*R_SF*	*FD_R_SF*
Implementation 1	PU_{Door}, SW_{FD_SF}, Comm	PU_{Door}, SW_{R_SF}	PU_{Cam}, $SW_{FD\_R\_SF}$, Comm
Implementation 2	No redundancy	PU_{Cam}, SW_{R_SF}, Comm	PU_{Door}, $SW_{FD\_R\_SF}$, Comm

The HW/SW architecture in Table 4 displays the implementation of the health management configuration in Table 9 for the different subfunctions with redundancies of the door status control main function.

Figure 12 and Table 10 show respectively the relative failure probability and relative cost of different HW/SW architectures for alternative redundancy strategies displayed in Table 8 normalized with the architecture without redundancies (cf. Table 2).

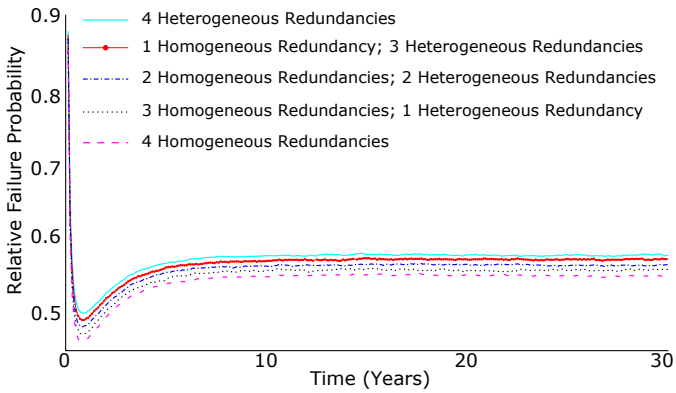

Figure 12: Relative failure probability of door status control configurations.

Table 10: Normalized cost of door status control configurations.

Conf.	HW, SW, Comm Cost	Downtime Cost
#1	1.221	0.583
#2	1.248	0.576
#3	1.281	0.57
#4	1.308	0.562
#5	1.2903	0.556

The following improvements have been observed at T=20 years with respect to the configuration without redundancies (cf. Figure 12): (#1): 42% better; (#2): 42.57% better; (#3): 43.23% better; (#4) 44.07% better; and (#5): 44.74% better. When considering the cost of hardware, software and communication implementations, heterogeneous redundancy configurations are cheaper than homogeneous redundancy configurations. However, with downtime costs, the less reliable the architecture, the higher its cost. Accordingly, heterogeneous redundancy configurations are more expensive than homogeneous redundancy configurations.

To examine the influence of reconfiguration strategies we have evaluated the failure probability for input, control, and output subfunctions with different reconfiguration arrangements for input subfunction implementations. Table 11 displays the arrangement of reconfigurations, where the subscript indicates the priority of the software reconfiguration implementation. All these configurations have the same fault detection configuration displayed in Table 9.

The system failure probability does not vary changing the number and distribution of reconfiguration implementations. However, focusing on Eq. (6) and Eq. (11) there are some properties worth mentioning. Taking door closed detection subfunction as a reference (note that the remainder of input subfunctions are characterized equally — door open detection, obstacle detection and door velocity), Table 12 shows the failure probability of the reconfiguration sequence failure event ($\mathcal{F}_{\text{R.Seq-DCD}}$ — Eq. (6)) and the reconfiguration subfunction failure event ($\mathcal{F}_{\text{R\_DCD}}$ — Eq. (11)) at T=10 years. These events have been analysed for different values of failure rates for health management software implementations (fault detection, reconfiguration, and reconfiguration's fault detection): SW_FD, SW_R, SW_FD_R. We have modified the failure rates of these software resources altogether (denoted collectively as λ_{SW_HM}) to see the effect on the failure probability.

Table 11: Reconfiguration distribution strategies.

Conf.	*Reconfiguration Implementation Distributions*
1R	**PU$_{\text{Door}}$**(R_DOD$_1$, R_DCD$_1$, R_OD$_1$, R_DV$_1$)
2RC	**PU$_{\text{Door}}$**(R_DOD$_1$, R_DCD$_1$,R_OD$_1$, R_DV$_1$); **PU$_1$**(R_DOD$_2$, R_DCD$_2$, R_OD$_2$, R_DV$_2$)
2RD	**PU$_{\text{Door}}$**(R_DOD$_1$, R_DCD$_2$); **PU$_1$**(R_DOD$_2$, R_DCD$_1$); **PU$_2$**(R_OD$_1$, R_DV$_2$); **PU$_3$**(R_OD$_2$, R_DV$_1$)
3RC	**PU$_{\text{Door}}$**(R_DOD$_1$, R_DCD$_1$, R_OD$_1$, R_DV$_1$); **PU$_1$**(R_DOD$_2$, R_DCD$_2$, R_OD$_2$, R_DV$_2$); **PU$_2$**(R_DOD$_3$, R_DCD$_3$, R_OD$_3$, R_DV$_3$)
3RD	**PU$_{\text{Door}}$**(R_DOD$_1$,R_DCD$_2$,R_OD$_3$); **PU$_1$**(R_DOD$_2$,R_DCD$_1$,R_DV$_3$); **PU$_2$**(R_DOD$_3$,R_OD$_1$,R_DV$_2$); **PU$_3$**(R_DCD$_3$,R_OD$_2$,R_DV$_1$)

Table 12: Reconfiguration events failure probability.

Events	**1R**	**2RD**	**2RC**	**3RD**	**3RC**
$\mathcal{F}_{\text{R.Seq-DCD}}(\lambda_{SW\_HM} = 0.05)$	0.013	0.005	0.005	0.003	0.004
$\mathcal{F}_{\text{R.Seq-DCD}}(\lambda_{SW\_HM} = 0.15)$	0.014	0.008	0.009	0.007	0.007
$\mathcal{F}_{\text{R.Seq-DCD}}(\lambda_{SW\_HM} = 0.25)$	0.016	0.011	0.011	0.009	0.009
$\mathcal{F}_{\text{R\_DCD}}(\lambda_{SW\_HM} = 0.05)$	0.312	0.138	0.140	0.124	0.127
$\mathcal{F}_{\text{R\_DCD}}(\lambda_{SW\_HM} = 0.15)$	0.571	0.313	0.316	0.274	0.275
$\mathcal{F}_{\text{R\_DCD}}(\lambda_{SW\_HM} = 0.25)$	0.761	0.466	0.466	0.390	0.391

The following characteristics are identified in Table 12:

- As the number of redundant implementations of reconfiguration increase, the failure probability of $\mathcal{F}_{R.Seq.\_SF}$ and $\mathcal{F}_{R\_SF}$ decreases.
- As the failure rate of the health management implementations increases, the failure probability of $\mathcal{F}_{R.Seq.\_SF}$ and $\mathcal{F}_{R\_SF}$ also increase.
- $\mathcal{F}_{R.Seq.\_SF}$ is lower than $\mathcal{F}_{R\_SF}$ due to the sequence-dependent constraint (cf. Eq. (6)).

Taking the HW/SW architecture with the redundancy configuration #1 as reference configuration (see Table 8), the influence of fault detection, reconfiguration and communication implementations

have been analysed assuming their ideal and real behaviour. Figure 13 shows the failure probability of these configurations.

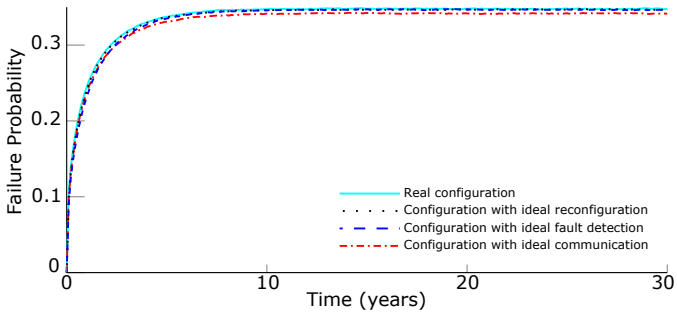

Figure 13: Door status control failure probability with ideal assumptions.

Table 13: Figure 13 values at T=15 years.

Configuration	*Failure Probability*
Real Configuration	0.348
Ideal Reconfiguration	0.347
Ideal Fault Detection	0.347
Ideal Communication	0.342

Figure 13 shows that the influence of the communication is more important than health management implementations because the communication influences many subfunctions and implementations at the same time. In this case, there is no difference in the influence of fault detection and reconfiguration implementations and their influence can be considered negligible (cf. Table 13).

7 Conclusions and Future Work

In this paper we have extended the recently proposed D3H2 methodology to model and evaluate repairable systems for the cost-effective design of dependable reconfigurable systems. Prioritized repair strategies are taken into account including components with complex logic and repeated events. The compositional modelling in D3H2 improves traceability between design and dependability models.

Application of the method to a railway case study has confirmed that the reuse of system resources reduces system cost compared with the addition of extra hardware components. However, this is only true when the additional cost incurred from increased failure probability of the system is not greater than the extra cost of homogeneous redundancy. When excluding downtime costs, heterogeneous redundancies are cheaper than homogeneous redundancies. However, downtime cost is higher with less reliable architectures and it is more penalising than hardware, software, and communication costs. D3H2 assists in the trade-off analysis between these properties and it enables informed decision making. The D3H2 methodology also includes the effect of health management mechanisms on system dependability. It is true that in many cases their effect may not be significant for the system performance, but assuming them ideal may result in an optimistic system evaluation. Therefore, their effect needs to be evaluated, specially for safety-critical systems.

When evaluating reconfiguration strategies, distributed reconfiguration strategies have shown a lower failure probability than the centralised reconfiguration redundancies in the analysed case study. However, it should be noted that the effect of increasing reconfiguration redundancies on system failure probability is attenuated because there are sequence-dependent intermediate, lower-level failure events. That is, the failure of the reconfiguration subfunction occurs when first the reconfiguration mechanism fails and then the subfunction implementation failure occurs. This time-dependent condition constraints the effect of increasing reconfiguration redundancies on the system failure probability.

As shown in the case study, optimisation of design decisions with respect to the level and type of redundancy and reconfiguration strategies to maximize dependability and minimize the cost are feasible within the D3H2 methodology. We acknowledge that the methodology assumes a design rationale and process, which designers may not wish to use in every application. However, the innovative and useful aspects of D3H2 such as dependability modelling can be adopted within other design methods. Our future goals towards improving D3H2 will focus on improving the proposed approach by addressing the following extensions:

- Automatic extraction of the dependability evaluation models: this approach would alleviate modelling errors (e.g., using meta-modelling techniques[47]) and accordingly enable the implementation of meta-heuristics, e.g., extending the work in[2,13] to automate and optimise design decisions. One possible direction is synthesis of D3H2 with model-based dependability analysis techniques[13].

- Formal identification of heterogeneous redundancies: this is a challenging task for complex systems because there may not be a deterministic relationship between variables. Further refinement of the proposed identification approach could focus on formalising engineering knowledge or exploring multi-physics based modelling formalisms[48].

- Verification of heterogeneous redundancies: include architecture-specific requirements such as timeliness constraints[49] or memory and processing capacity.

- Quality degradation caused by the use of heterogeneous redundancies: analyse other properties than the failure probability.

- Repair and maintenance strategies: the train operates through different phases and it is possible to schedule repair and maintenance actions accordingly. For instance, if an asset is not critical, it can be left in the failed state until reaching a railway depot and repair altogether. For critical assets, condition-based maintenance techniques[50] can be considered to monitor the condition of components and schedule maintenance before their failure occurrence reducing downtime costs[51].

- Application of the D3H2 methodology at the overall system level including interactions and dependencies between all the system main functions through high level functions.

Acknowledgements

This work was partially funded by Mondragon University and CAF Power & Automation company. The authors would like to thank the reviewers for their valuable comments that helped to improve the clarity and completeness of the paper and also colleagues at CAF Power & Automation for the discussions that helped to develop the case study.

References

1. Elegbede A, Chu C, Adjallah K, Yalaoui F. Reliability allocation through cost minimization. *Reliability, IEEE Transactions on* March 2003; **52**(1):106–111.

2. Dhouibi MS, Saintis L, Barreau M, Perquis JM. Safety driven optimization approach for automotive systems. *2015 Annual Reliability and Maintainability Symposium (RAMS)*, 2015; 1–7, doi:10.1109/RAMS.2015.7105113.

3. Han J, Gao J, Jonker P, Qi Y, Fortes JAB. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Design Test of Computers* July 2005; **22**(4):328–339, doi:10.1109/MDT.2005.97.

4. Aizpurua JI, Muxika E. Functionality and dependability assurance in massively networked scenarios. *Safety, Reliability and Risk Analysis: Beyond the Horizon*, CRC Press, 2013; 1763 – 1771.

5. Aizpurua JI, Muxika E, Manno G, Chiacchio F. Heterogeneous redundancy analysis based on component dynamic fault trees. *Proceedings of PSAM 12*, 2014.

6. Aizpurua JI. Functionality and dependability assurance in massively networked scenarios. PhD Thesis, Electronics and Computing Department, Mondragon University January 2015.

7. Aizpurua JI, Muxika E, Papadopoulos Y, Chiacchio F, Manno G. Application of the D3H2 methodology for the cost-effective design of dependable systems. *Safety* 2016; **2**(2):9, doi:10.3390/safety2020009.

8. Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* 2004; **1**:11–33, doi:http://dx.doi.org/10.1109/TDSC.2004.2.

9. Garg H, Sharma S. Stochastic behavior analysis of complex repairable industrial systems utilizing uncertain data. *ISA Transactions* 2012; **51**(6):752 – 762, doi:http://dx.doi.org/10.1016/j.isatra.2012.06.012.

10. Aizpurua JI, Muxika E. Model based design of dependable systems: Limitations and evolution of analysis and verification approaches. *International Journal on Advances in Security* 2013; **6**:12–31.

11. Baier C, Katoen JP. *Principles of Model Checking*. The MIT Press, 2008.

12. Katsaros P, Angelis L, Lazos C. Performance and effectiveness trade-off for checkpointing in fault-tolerant distributed systems. *Concurrency and Computation: Practice and Experience* 2007; **19**(1):37–63, doi:10.1002/cpe.1059.

13. Adachi M, Papadopoulos Y, Sharvia S, Parker D, Tohdo T. An approach to optimization of fault tolerant architectures using HiP-HOPS. *Softw. Pract. Exp.* 2011; .

14. Cauffriez L, Renaux D, Bonte T, Cocquebert E. Systemic modeling of integrated systems for decision making early on in the design process. *Cybernetics and Systems* 2013; **44**:1–22.

15. Chen D, Lönn H, Mraidha C, Papadopoulos Y, Reiser M, Servat D, Azevedo LS, Piergiovanni ST, Walker M. Automatic optimisation of system architectures using EAST-ADL. *SAFECOMP 2013 - Workshop ASCoMS (Architecting Safety in Collaborative Mobile Systems), Toulouse, France*, 2013.

16. Perez D, Mirandola R, Merseguer J. On the relationships between QoS and software adaptability at the architectural level. *Journal of Systems and Software* 2014; **87**:17.

17. Strigini L. Fault tolerance against design faults. *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, Diab H, Zomaya A (eds.). John Wiley & Sons, 2005; 213–241.

18. Blanke M, Hansen S, Blas MR. Diagnosis for control and decision support in complex systems. *Proceedings Volume from the Special International Conference on Complex Systems*, 2011; 89–101.

19. Shelton CP, Koopman P. Improving system dependability with functional alternatives. *Proc. of DSN'04*, IEEE, 2004; 295–304.

20. Wysocki J, Debouk R. Methodology for assessing safety-critical systems. *Int. Journal of Modeling and Simulations* 2007; **27**(2):99–106.

21. Adler R, Schneider D, Trapp M. Engineering dynamic adaptation for achieving cost-efficient resilience in software-intensive embedded systems. *Proc. of Engineering of Complex Computer Systems*, IEEE, 2010; 21–30, doi:10.1109/ICECCS.2010.12.

22. Bieber P, Noulard E, Pagetti C, Planche T, Vialard F. Preliminary design of future reconfigurable IMA platforms. *SIGBED Rev.* 2009; **6**(3).

23. Dugan J, Bavuso S, Boyd M. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Trans. on Reliability* 1992; **41**(3):363–377, doi:10.1109/24.159800.

24. Manno G, Chiacchio F, Compagno L, D'Urso D, Trapani N. Conception of repairable dynamic fault trees and resolution by the use of RAATSS, a matlab toolbox based on the ATS formalism. *Reliability Engineering & System Safety* 2014; **121**:250 – 262, doi:http://dx.doi.org/10.1016/j.ress.2013.09.002.

25. Distefano S, Puliafito A. Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees. *IEEE Transanctions on Dependable and Secure Computing* 2009; **6**(1):4–17.

26. Papadopoulos Y, Walker M, Parker D, Rüde E, Hamann R, Uhlig A, Grätz U, Lien R. Engineering failure analysis and design optimisation with HiP-HOPS. *Engineering Failure Analysis* 2011; **18**(2):590–608.

27. Edifor E, Walker M, Gordon N. *Quantification of Priority-OR Gates in Temporal Fault Trees*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; 99–110, doi:10.1007/978-3-642-33678-2_9.

28. Edifor E, Walker M, Gordon N. *Quantification of Simultaneous-AND Gates in Temporal Fault Trees*. Springer International Publishing: Heidelberg, 2013; 141–151, doi:10.1007/978-3-319-00945-2_13.

29. Bouissou M, Bon JL. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering &; System Safety* ; **82**(2):149 – 63.

30. Gulati R, Dugan JB. A modular approach for analyzing static and dynamic fault trees. *Reliability and Maintainability Symposium. 1997 Proceedings, Annual*, 1997; 57–63, doi:10.1109/RAMS.1997.571665.

31. Yevkin O. An improved modular approach for dynamic fault tree analysis. *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*, 2011; 1–5, doi:10.1109/RAMS.2011.5754437.

32. Kaiser B, Gramlich C, Forster M. State-event fault trees - a safety analysis model for software-controlled systems. *Reliability Eng. System Safety* 2007; **92**(11):1521–1537.

33. Codetta-Raiteri D. Integrating several formalisms in order to increase fault trees' modeling power. *Reliability Engineering & System Safety* 2011; **96**(5):534 – 544, doi:http://dx.doi.org/10.1016/j.ress.2010.12.027.

34. Piriou PY, Faure JM, Lesage JJ. Modeling standby redundancies in repairable systems as guarded preemption mechanisms. *Dependable Control of Discrete Systems (DCDS)*, 5, Cancun, Mexico, 2015; 147–153.

35. IEC. Train Communication Network, IEC 61375. *Technical Report* 2007.

36. Marca DA, McGowan CL. *SADT: Structured Analysis and Design Technique*. McGraw-Hill, Inc.: New York, NY, USA, 1987.

37. MathWorks. Matlab/Simulink. `http://www.mathworks.com`; Accessed: 25/04/2016 2015.

38. Sanders WH, Meyer JF. Lectures on formal methods and performance analysis. chap. Stochastic Activity Networks: Formal Definitions and Concepts, Springer, 2002; 315–343.

39. Meyer JF, Movaghar A, Sanders WH. Stochastic activity networks: Structure, behavior, and application. *International Workshop on Timed Petri Nets*, IEEE Computer Society: Washington, DC, USA, 1985; 106–115.

40. Di Martino C, Cinque M, Cotroneo D. Automated generation of performance and dependability models for the assessment of wireless sensor networks. *Computers, IEEE Transactions on* June 2012; **61**(6):870–884, doi:10.1109/TC.2011.96.

41. Giandomenico FD, Itria M, Masci P, Nostro N. Automated synthesis of dependable mediators for heterogeneous interoperable systems. *Reliability Engineering & System Safety* 2014; **132**:220 – 232, doi:http://dx.doi.org/10.1016/j.ress.2014.08.001.

42. Sanders WH, Meyer JF. Reduced base model construction methods for stochastic activity networks. *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, 1989; 74–84, doi:10.1109/PNPM.1989.68541.

43. Kanoun K. Real-world design diversity: a case study on cost. *Software, IEEE* Jul 2001; **18**(4):29–33, doi:10.1109/MS.2001.936214.

44. IAEA. Component reliability data for use in probabilistic safety assessment, IAEA-TECDOC-478. *Technical Report* 1988.

45. JVC Professional. `http://pro.jvc.com/`; Accessed: 25/04/2016.

46. Vinod G, Santosh T, Saraf R, Ghosh A. Integrating safety critical software system in probabilistic safety assessment. *Nuclear Engineering and Design* 2008; **238**(9):2392 – 2399.

47. Henderson-Sellers B. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software* 2011; **84**(2):301 – 313, doi:http://dx.doi.org/10.1016/j.jss.2010.10.025.

48. Fritzson P. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press, 2011.

49. Priesterjahn C, Steenken D, Tichy M. Timed hazard analysis of self-healing systems. *Assurances for Self-Adaptive Systems - Principles, Models, and Techniques*. 2013; 112–151, doi:10.1007/978-3-642-36249-1_5.

50. Aizpurua JI, Catterson V. Towards a methodology for design of prognostics systems. *Annual Conference of the Prognostics and Health Management Society*, vol. 6, 2015.

51. Aizpurua JI, Catterson VM, Chiacchio F, D'Urso D. A cost-benefit approach for the evaluation of prognostics-updated maintenance strategies in complex dynamic systems. *Proceedings of the European Safety & Reliability Conference (ESREL'16)*, Glasgow, UK, 2016.

Acronyms and Abbreviations

DCA	: Door Control Algorithm	FD_SF	: Fault Detection of the SF
DCD	: Door Closed Detection	FP	: False Positive
DM	: Door Manipulation	MF	: Main Function
DOC	: Door Open Command	O	: Omission
DOD	: Door Open Detection	OD	: Obstacle Detection
DSC	: Door Status Control	PL	: Physical Location
DV	: Door Velocity	PU	: Processing Unit
EDD	: Enable Door Driver	R	: Reconfiguration
EDP	: Enable Door Passenger	R_SF	: Reconfiguration of the SF
FD	: Fault Detection	SF	: Subfunction
FD_R_SF	: Fault Detection of the R_SF	TCMS	: Train Control and Monitoring System

Authors' Biographies

Jose Ignacio Aizpurua is a Research Associate within the Institute for Energy and Environment at the University of Strathclyde, Scotland, UK. He received his Eng., M.Sc., and Ph.D. degrees from Mondragon University (Basque Country, Spain) in 2010, 2012, and 2015 respectively. He was a visiting researcher in the Dependable Systems Research group at the University of Hull (UK) during autumn

2014. His research interests include prognostics and health management, dependability, condition monitoring, and systems engineering.

Yiannis Papadopoulos is a professor and leader of the Dependable Systems research group at the University of Hull. He pioneered the HiP-HOPS MBSA method and contributed to the EAST-ADL automotive design language, working with Volvo, Honda, Continental, Honeywell, and DNV-GL, among others. He is actively involved in two technical committees of IFAC (TC 1.3 & 5.1).

Eñaut Muxika is a lecturer and a researcher at Mondragon Unibertsitatea and he obtained his PhD in Electrical Engineering from the Institute National Polytechnique de Grenoble (INPG) in 2002. He has worked in machine-tool and power electronics control systems. His current research interests include reliability, availability, safety and performance modeling, model-based system engineering and adaptive hardware, software, communication system design.

Ferdinando Chiacchio is a Researcher in the Department of Electrical Electronic and Computer Engineering at the University of Catania. He received his Laurea and Ph.D. degrees from University of Catania in 2005 and 2010 respectively. His research areas concern reliability, performability, communication protocols for home and industrial control and automation, HPC computing and immunomics.

Gabriele Manno is Senior Researcher at DNV GL in the Strategic Research and Innovation department. He received his Bachelor, M.Sc., and Ph.D. degrees from the University of Catania and a MSc in Business Administration from IlSole24Ore Business School. His interests include dependability theory and advanced prognostics as well as digitalization, big data and industrial platforms with specific focus on the shipping industry.